

Figures

TABLE 10-1

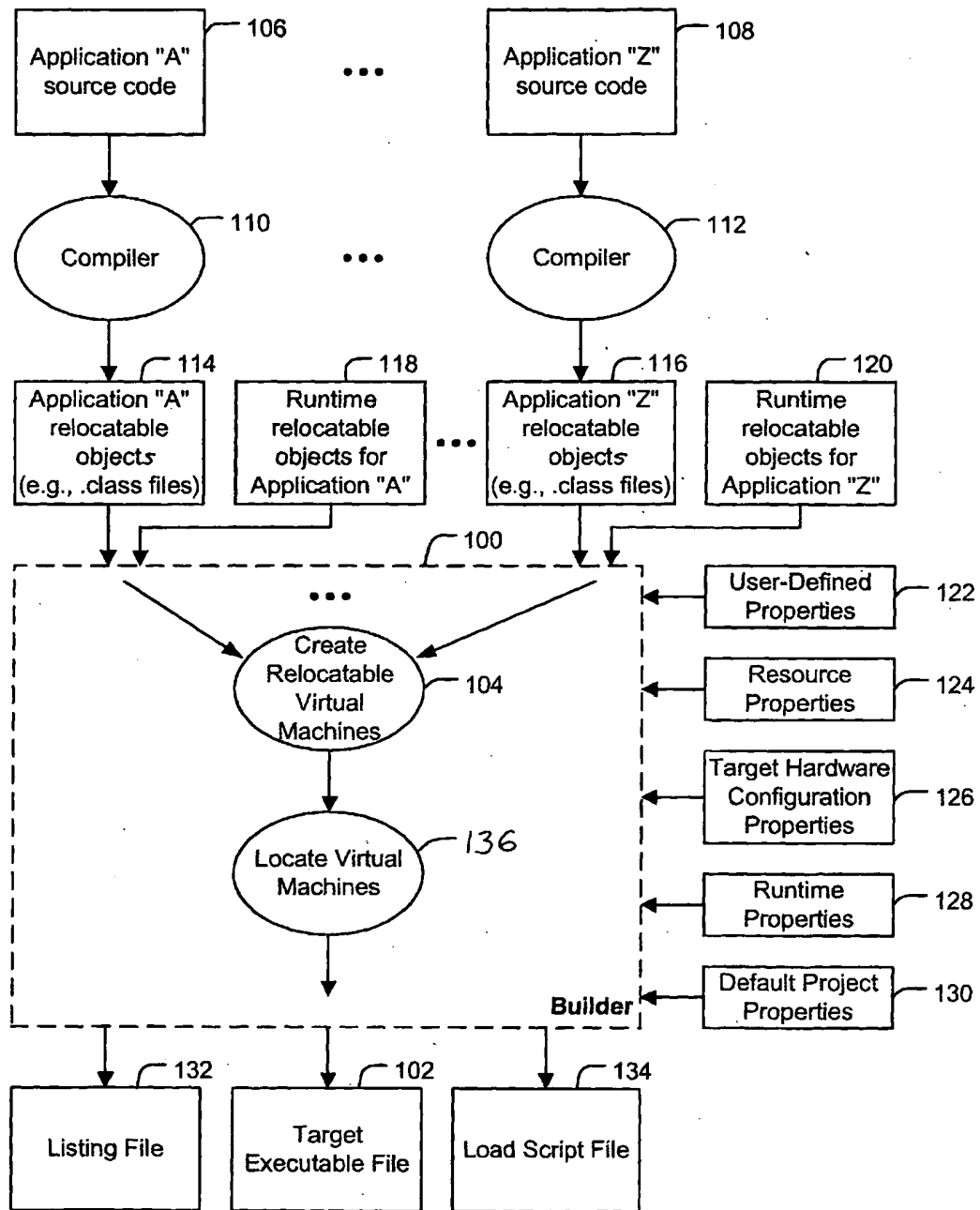


Fig. 1

```
graph TD
    200{User-Defined Parameter?} -- Y --> 202[Return User-Defined Parameter]
    200 -- N --> 204{Any installed resources?}
    204 -- Y --> 206{Does any installed Resource specify the Parameter?}
    204 -- N --> 210{Do Target HW configuration properties specify the parameter?}
    206 -- Y --> 208[Return Resource Specified Parameter]
    206 -- N --> 210
    210 -- Y --> 216[Return HW Configuration Specified Parameter]
    210 -- N --> 218{Do Runtime properties specify the parameter?}
    218 -- Y --> 220[Return Runtime Specified Parameter]
    218 -- N --> 222[Return a default value for the parameter]
```

Flowchart illustrating a process for returning a parameter value based on various checks:

- Decision 200: User-Defined Parameter?
 - If Yes (Y), proceed to 202: Return User-Defined Parameter.
 - If No (N), proceed to 204.
- Decision 204: Any installed resources?
 - If Yes (Y), proceed to 206.
 - If No (N), proceed to 210.
- Decision 206: Does any installed Resource specify the Parameter?
 - If Yes (Y), proceed to 208: Return Resource Specified Parameter.
 - If No (N), proceed to 210.
- Decision 210: Do Target HW configuration properties specify the parameter?
 - If Yes (Y), proceed to 216: Return HW Configuration Specified Parameter.
 - If No (N), proceed to 218.
- Decision 218: Do Runtime properties specify the parameter?
 - If Yes (Y), proceed to 220: Return Runtime Specified Parameter.
 - If No (N), proceed to 222: Return a default value for the parameter.

Fig. 2

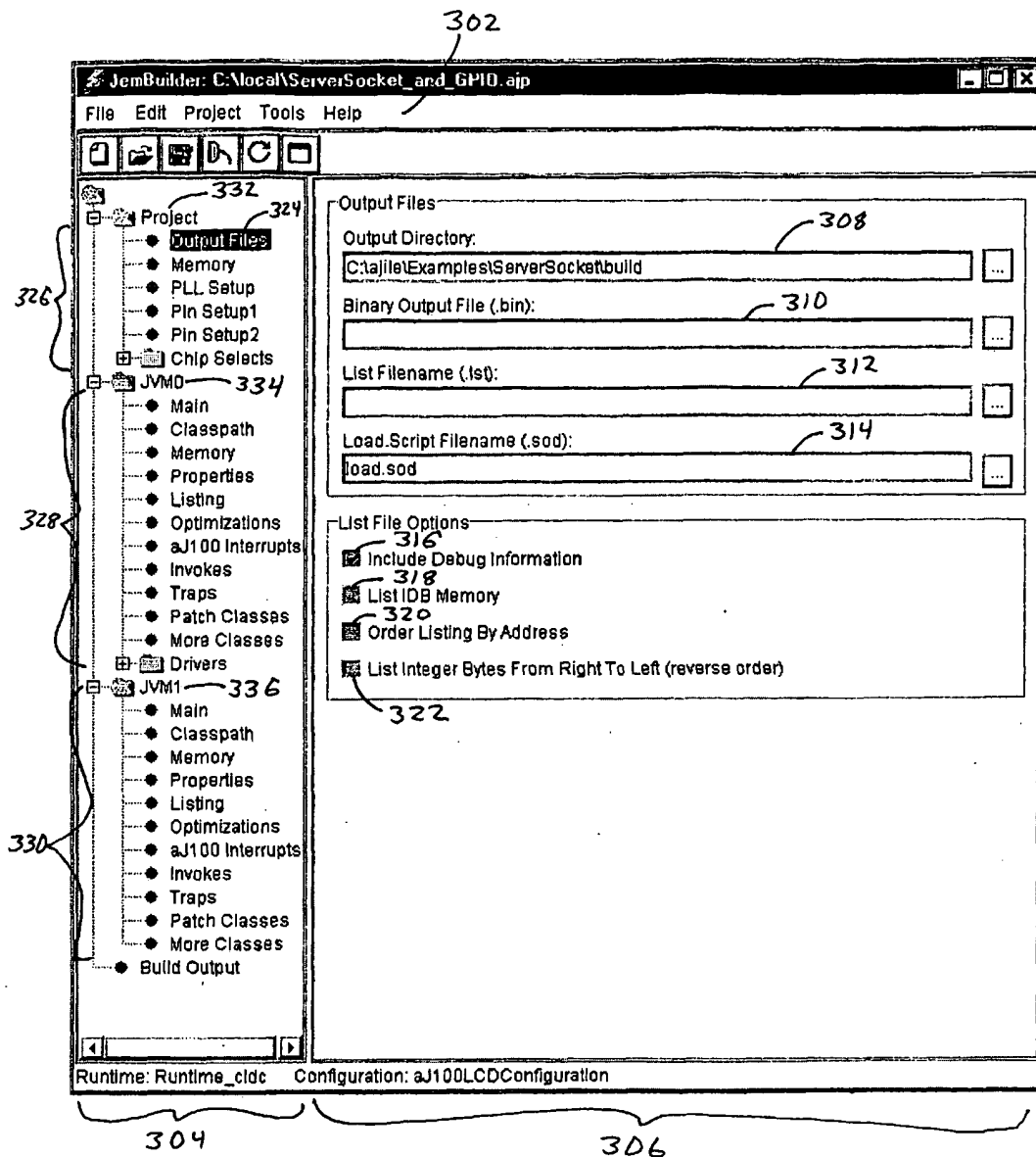


Fig. 3

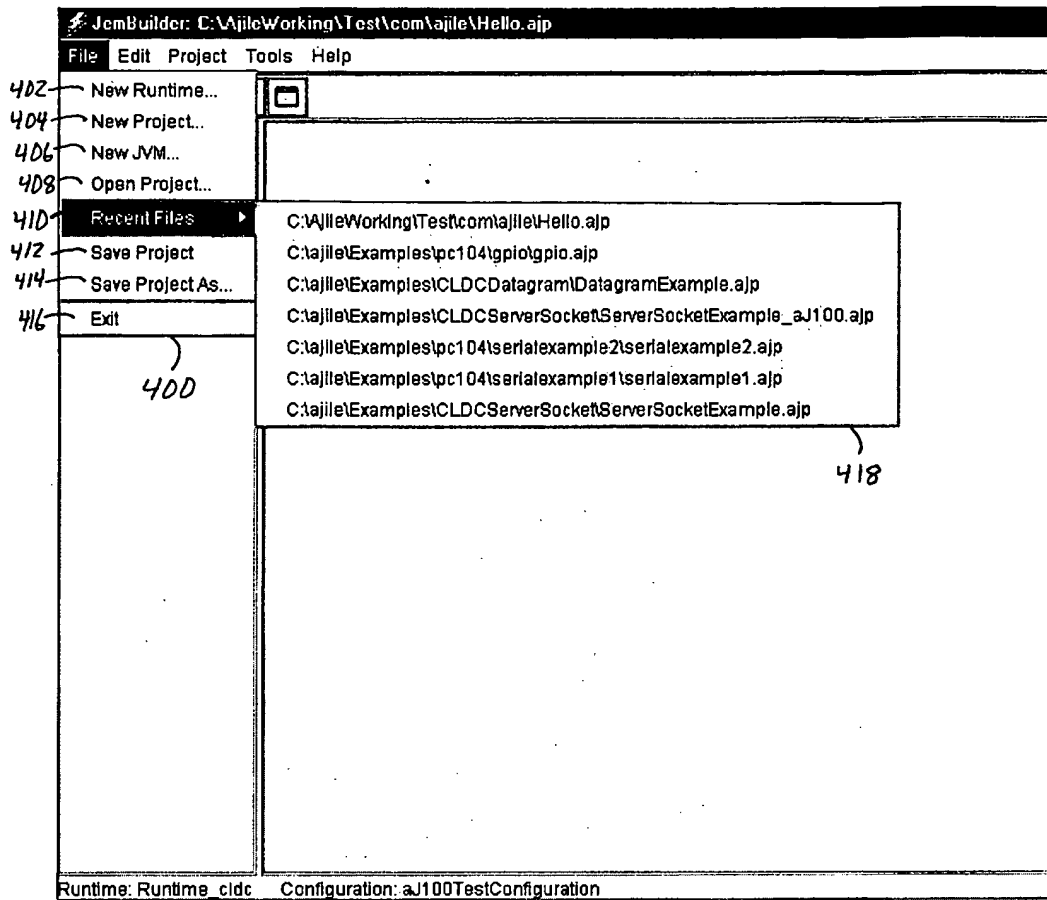


Fig. 4

Downloaded from www.scribd.com

New JVM step 1 of 4

Name

A Project can contain 1 or more JVM's. Each JVM runs independently, has it's own memory, resources, and main method.
Each JVM has a name that is referenced in the navigation tree
Example: SystemJvm
Enter the name of the new JVM:

JVM2

Next > Cancel

502

500

Fig. 5

New JVM step 2 of 4

Main Class Name

Each JVM contains one entry point or main method. The package name must be specified followed by a '.' and the class name. This is called the fully qualified class name.
Example: com.ajile.example.Example

Enter the fully qualified class name:

GPIONonitor

< Previous Next > Cancel

602

600

Fig. 6

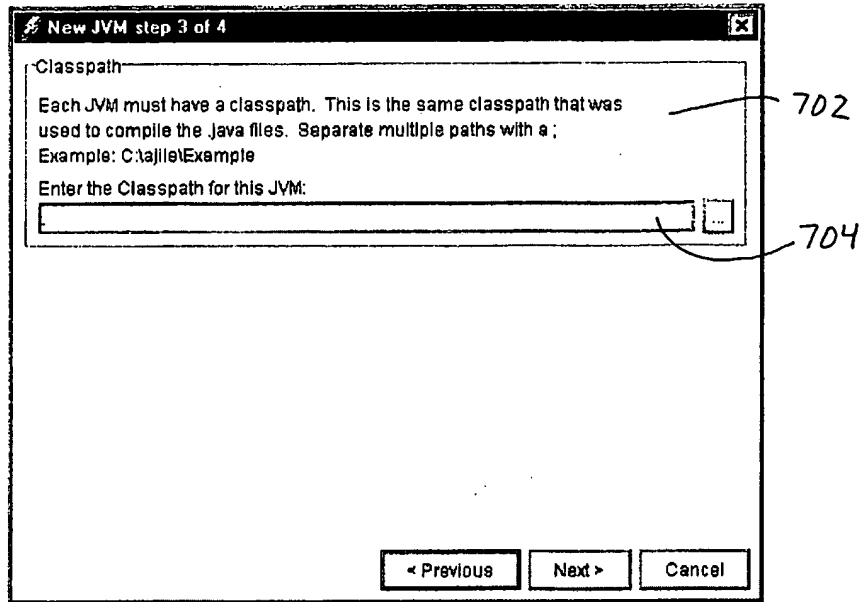


Fig. 7

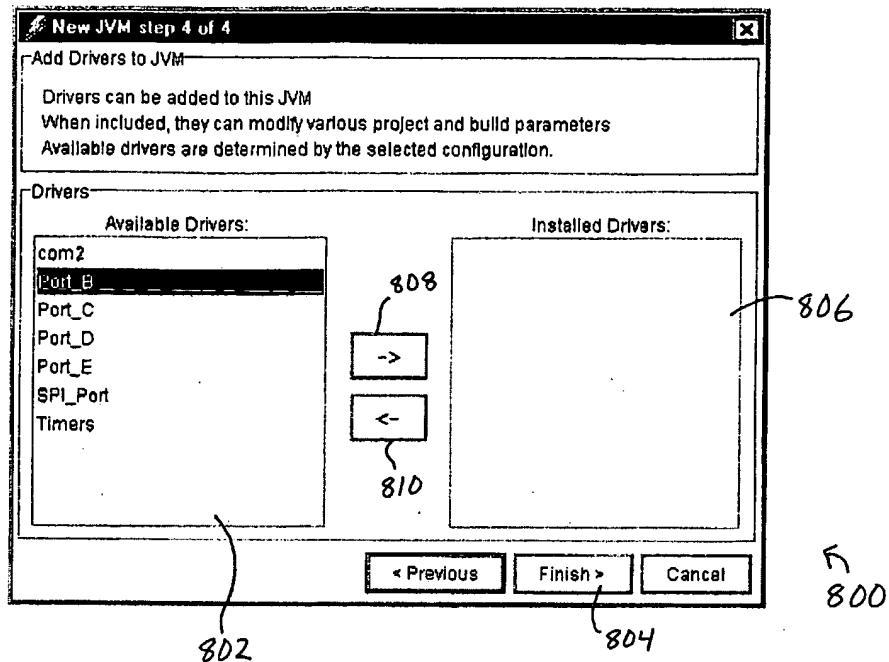


Fig. 8

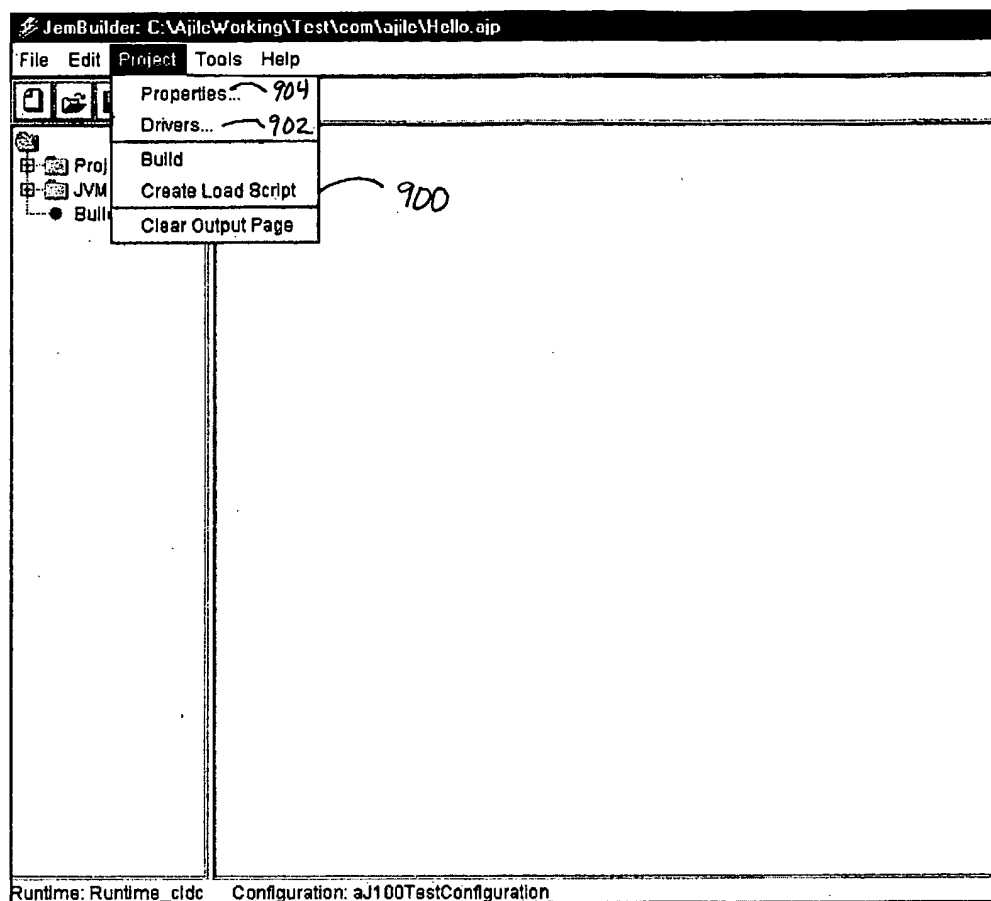


Fig. 9

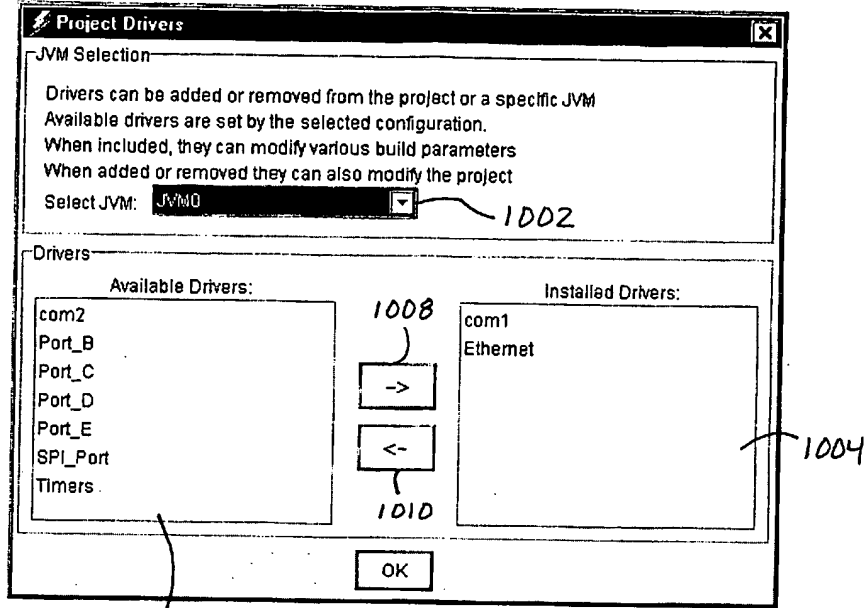


Fig. 10

~1000

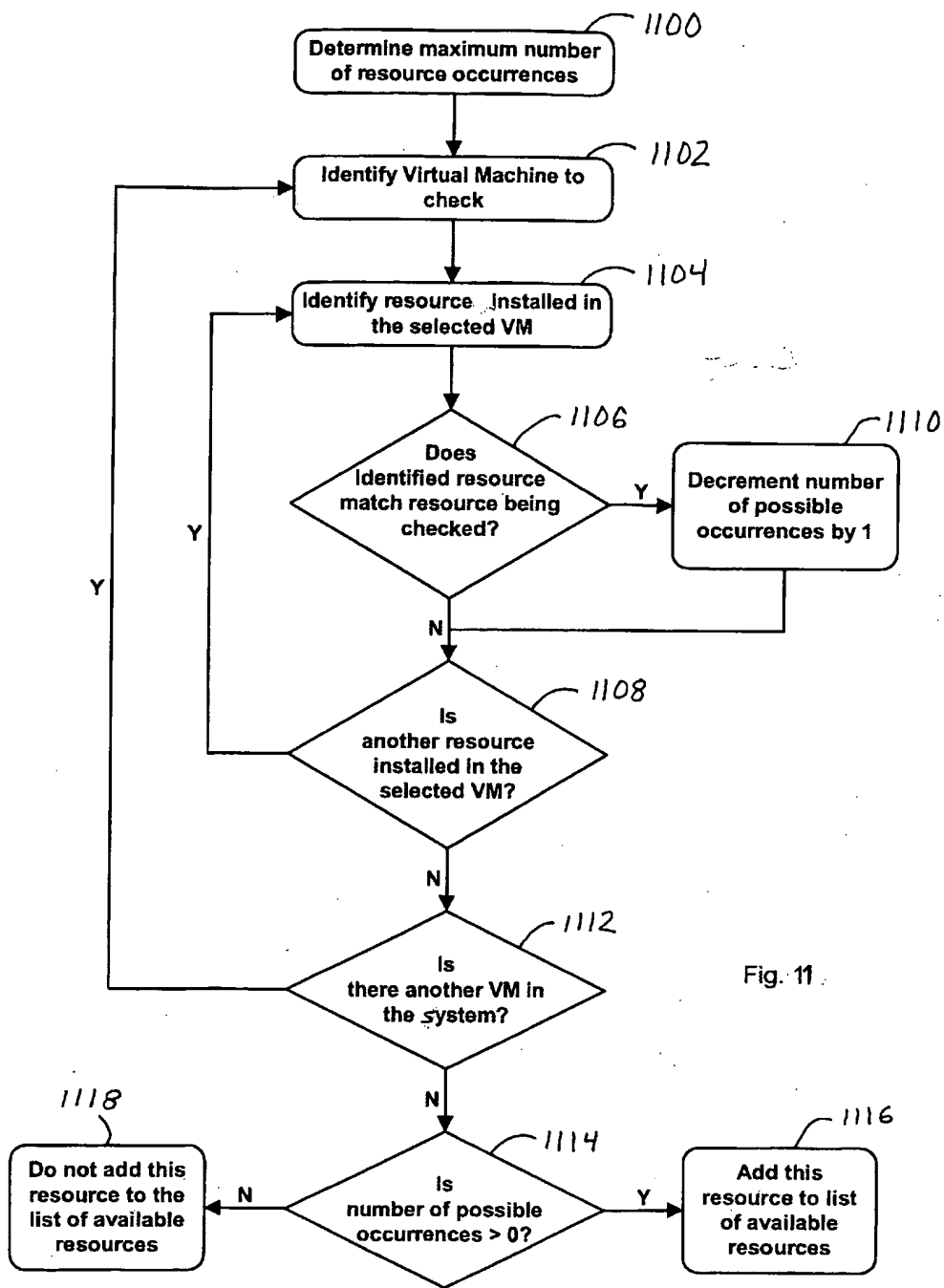


Fig. 11

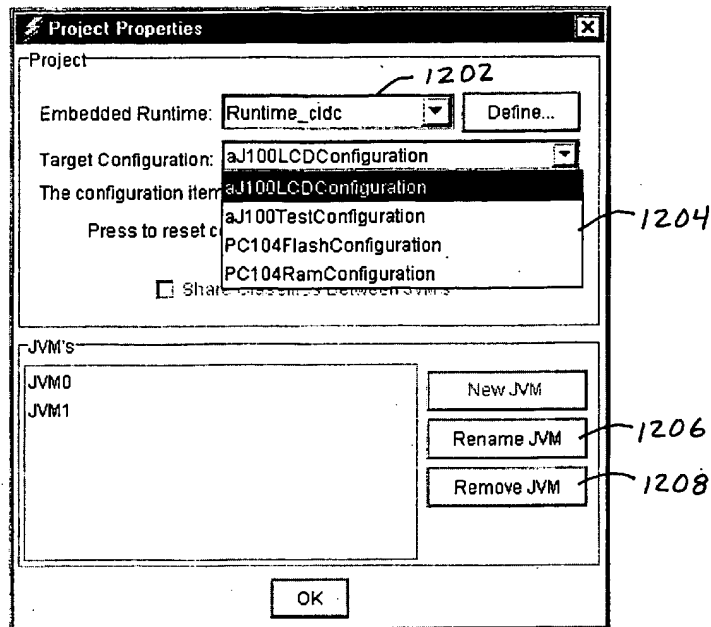


Fig. 12

1200

140020163560

JemBuilder: C:\local\ServerSocket_and_GPIO.ajp

File Edit Project Tools Help

Project

- Output Files
- Memory
- PLL Setup
- Pin Setup1
- Pin Setup2
- Chip Selects
- JVM0
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aj100 Interrupts
 - Invokes
 - Traps
 - Patch Classes
 - More Classes
- Drivers
- JVM1
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aj100 Interrupts
 - Invokes
 - Traps
 - Patch Classes
 - More Classes
- Build Output

Phased Locked Loop Enable

☒ Enable PLL operation

The lock timeout can be used to bypass the PLL for a number of clock cycles, until the PLL stabilizes. The input clock will be used for the specified number of input clocks, then a switch is made to the PLL output. If the lock timeout is disabled, the PLL output will be used all of the time.

3,670,016 Number of Cycles to PLL Driven Clock

Clock Frequency Calculation

10000000 Enter your input Clock Frequency in Hz

10 Select the PLL Multiplier

1 Select the PLL Divider

100000000 Internal Clock Frequency in Hz

49 Internal Time Prescaler

Clock Out Frequency

The Clock Out pin can be disabled, or its frequency divided down

2 Select the Clock Output Divider

50000000 Clock Out Frequency in Hz

Runtime: Runtime_cldc Configuration: aj100LCDConfiguration

1402

Fig.14

1400

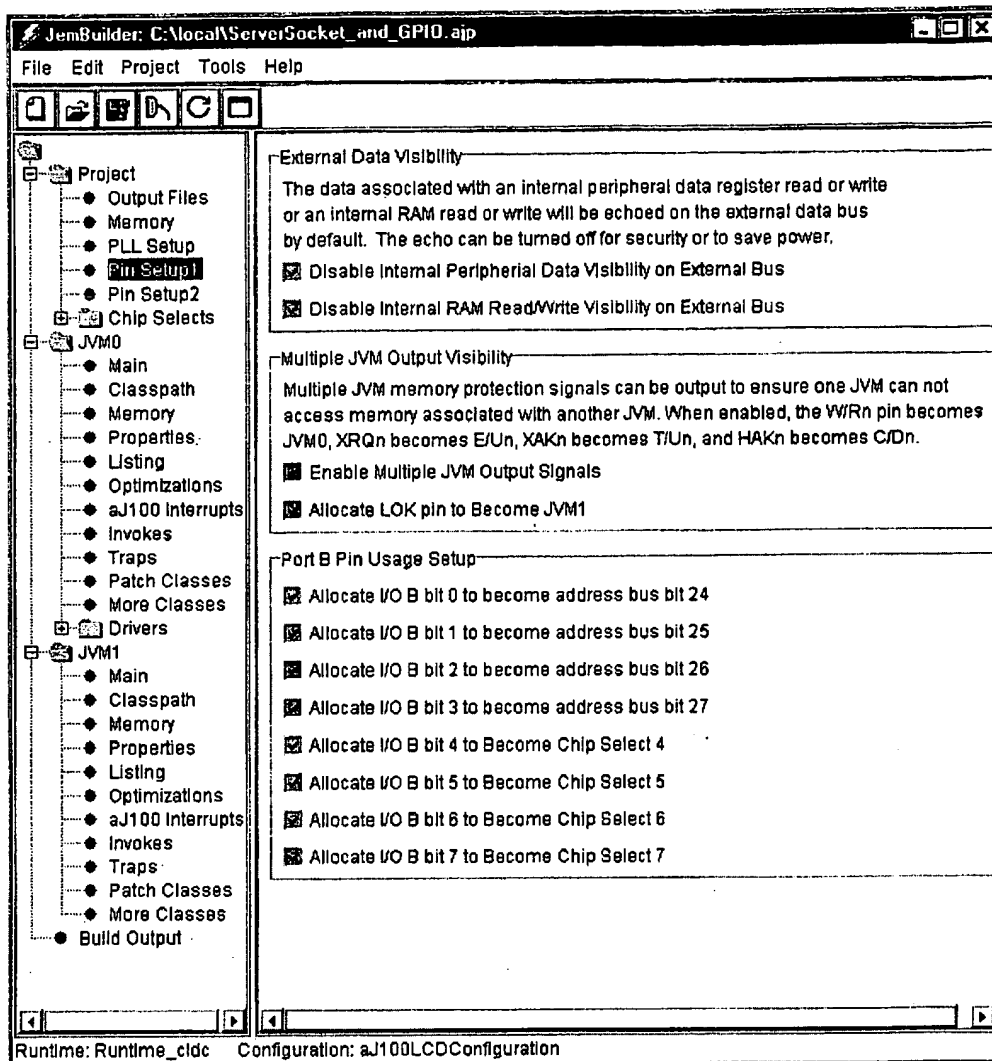


Fig. 15

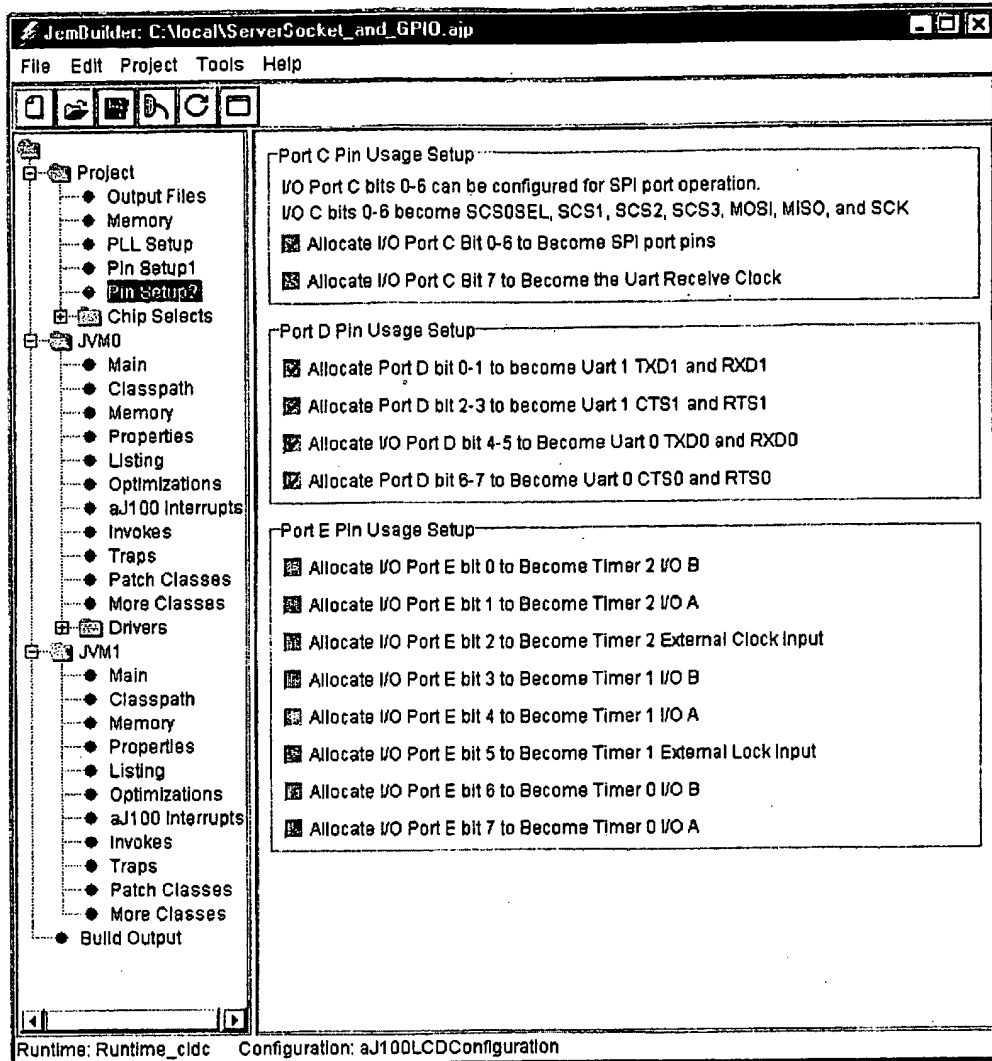


Fig. 16

1700: 09213950

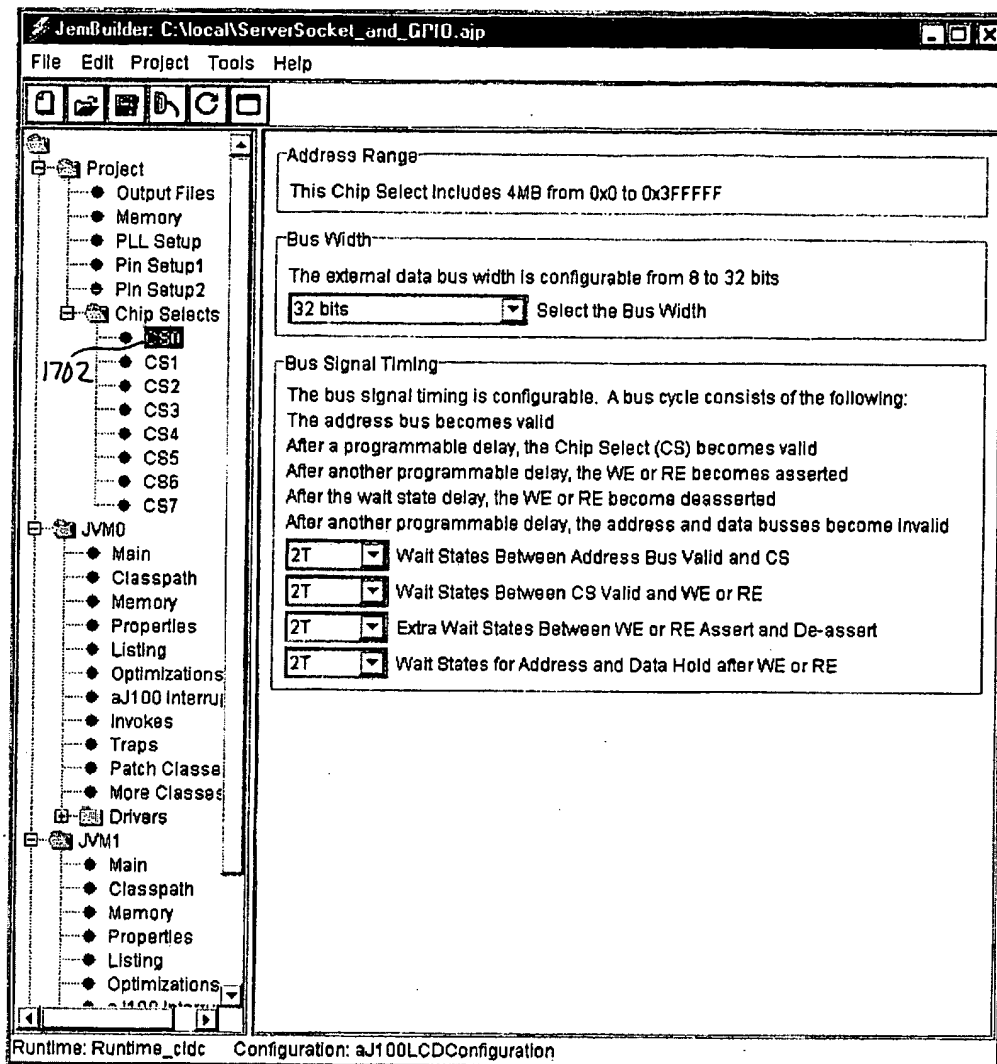


Fig. 17

1700

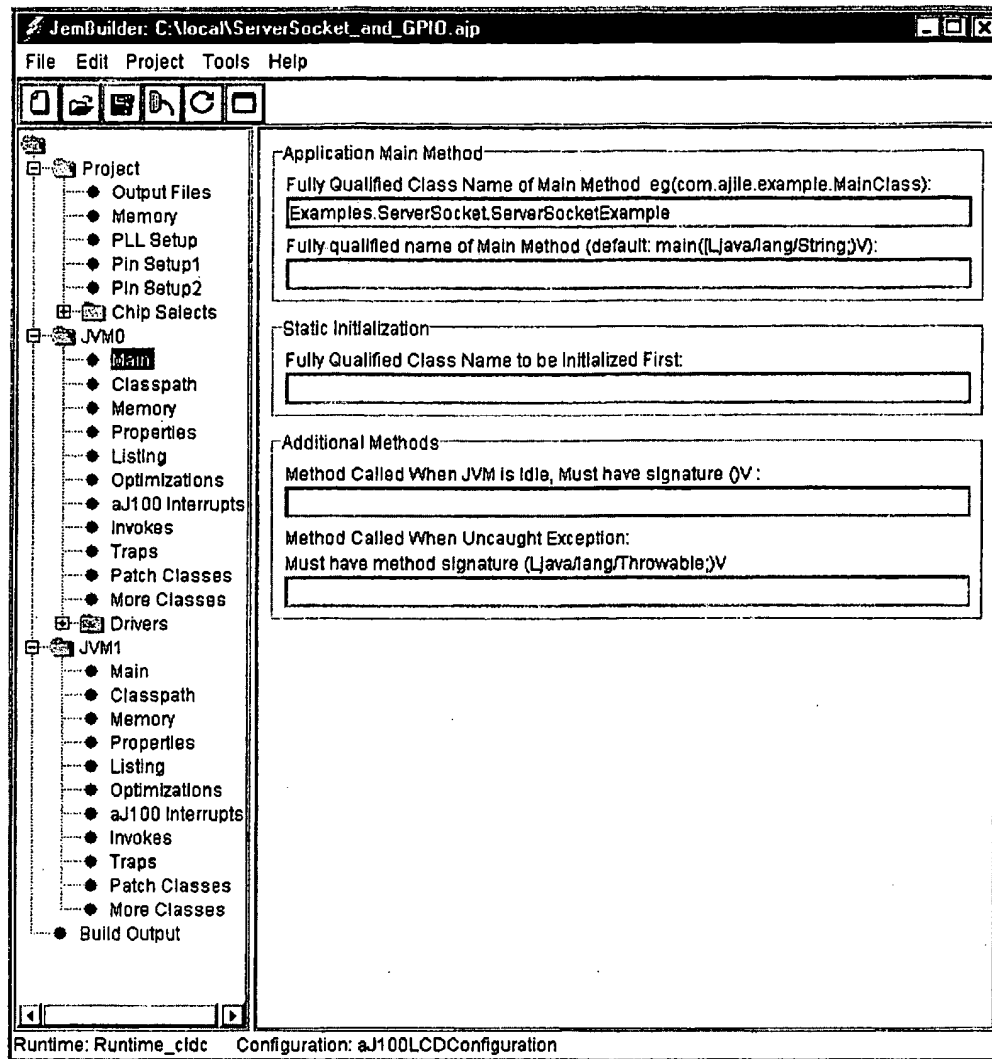


Fig. 18

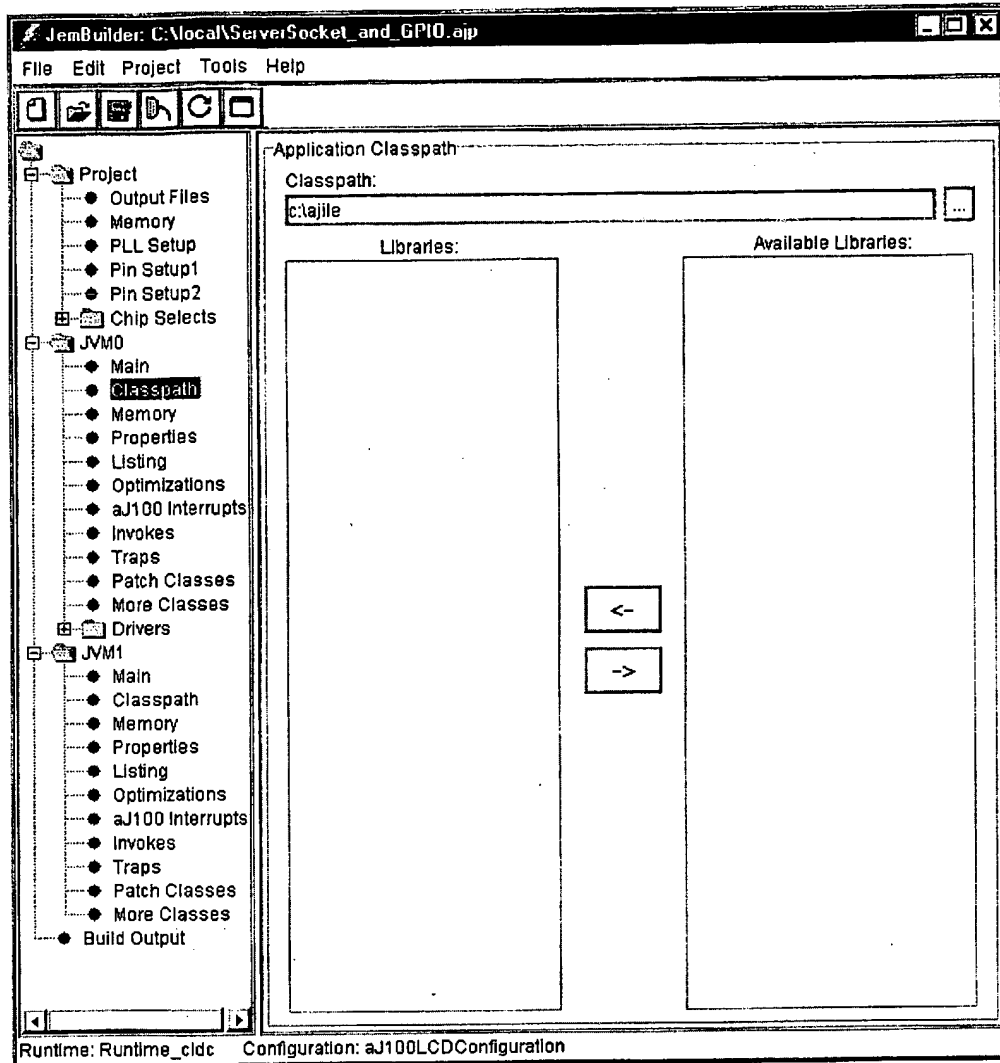


Fig. 19

106060-63218960

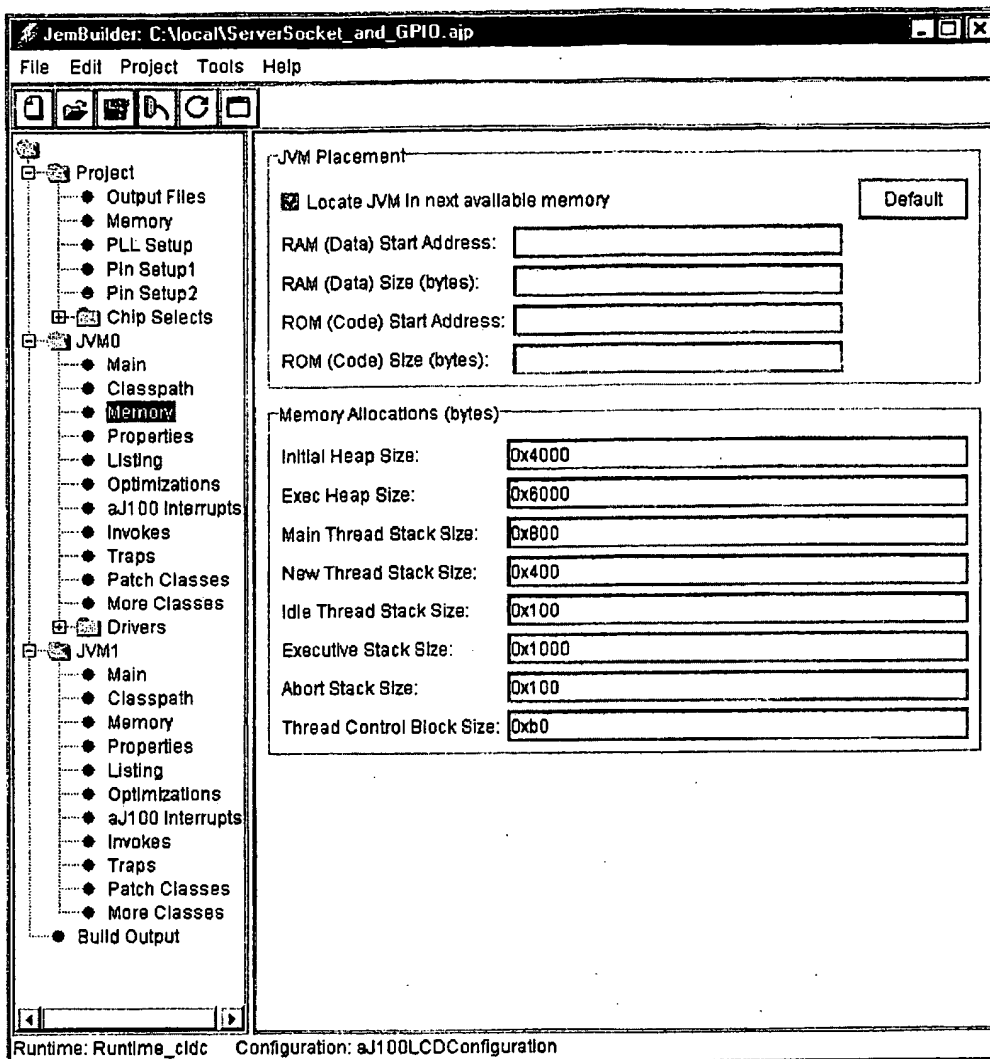


Fig. 20

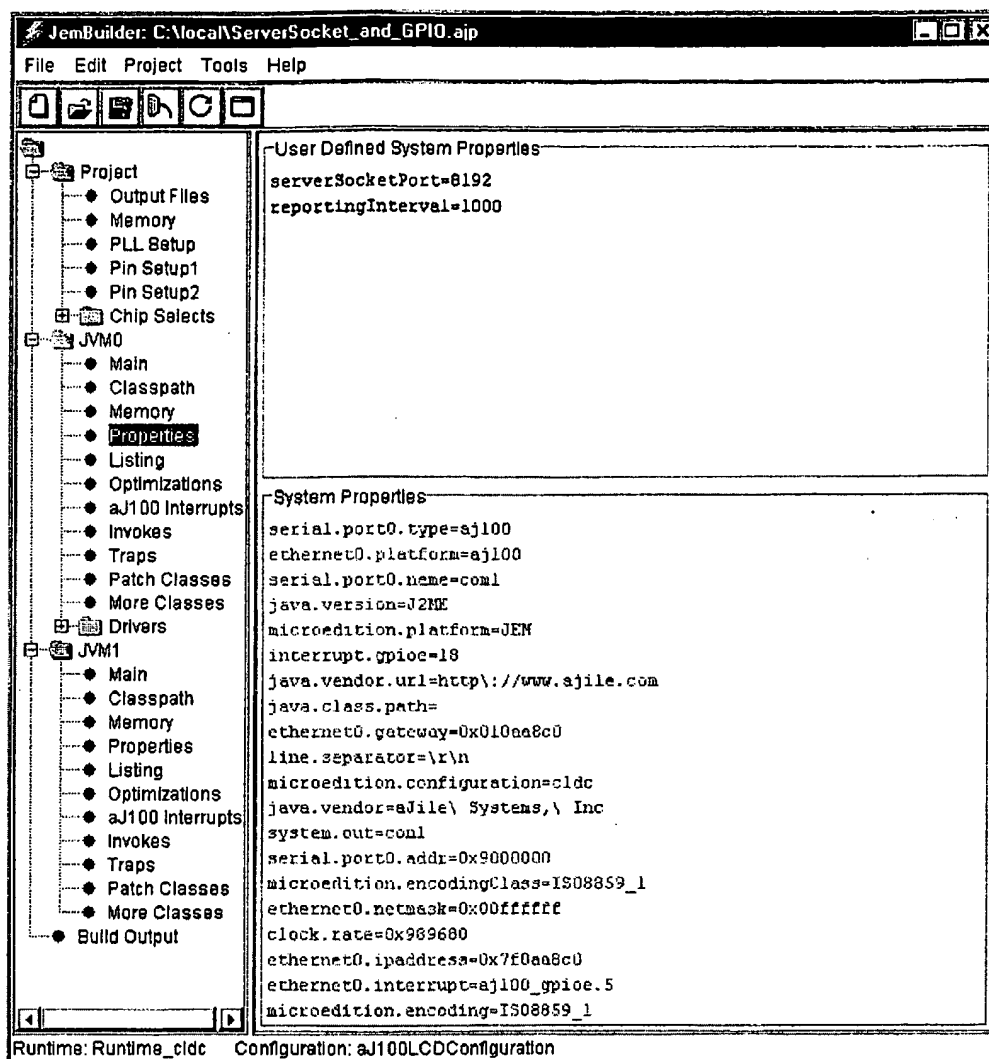


Fig. 21

T02060" 63213300

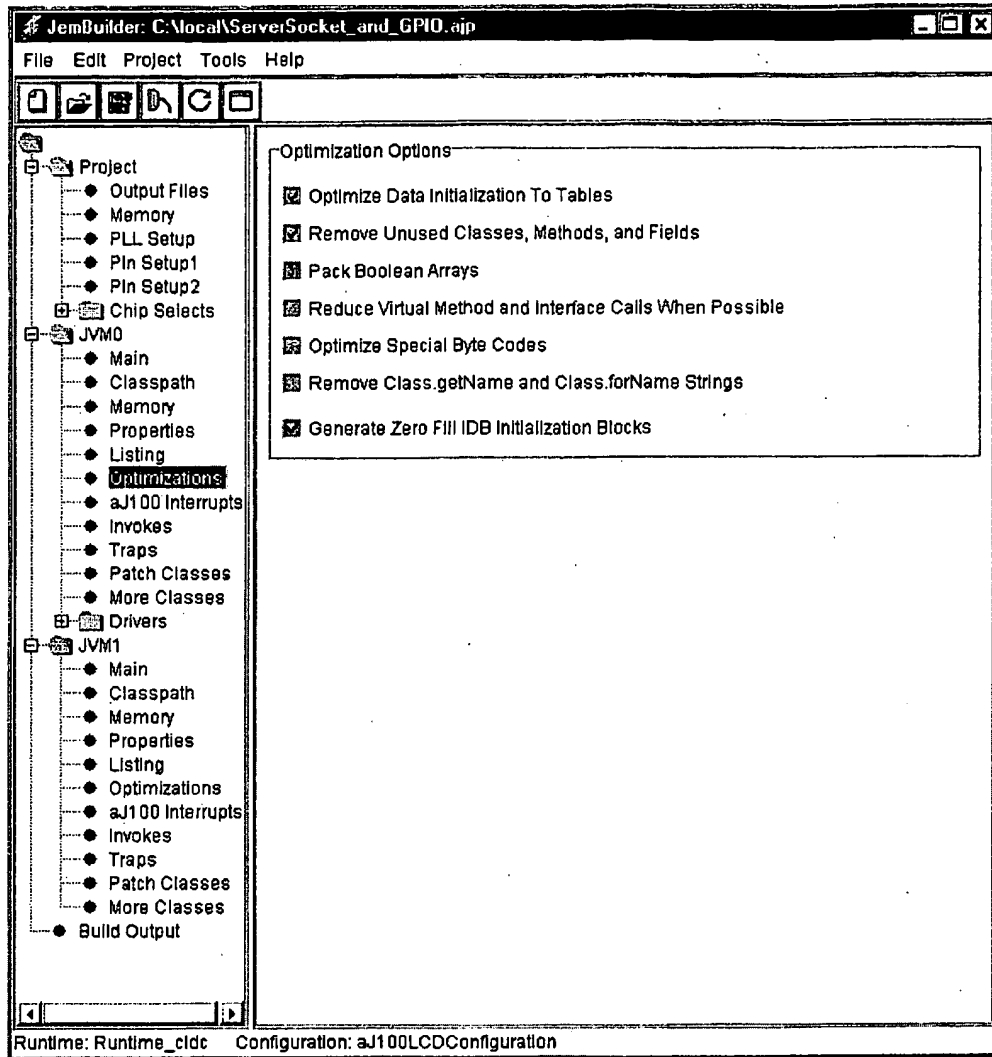


Fig. 23

The screenshot shows the JemBuilder IDE interface. On the left is the Project Explorer, displaying a tree structure with 'Project' and 'JVM0' folders. Under 'JVM0', the 'aj100 Interrupts' option is selected. On the right is the 'Transfer Error XERR' configuration window, which contains various interrupt settings. The 'Peripheral Interrupts' section is expanded, showing a table with columns for 'Timer', 'Priority', and 'Name'. The 'aj100 Interrupts' option is selected in the Project Explorer.

Timer:	Priority:	
	10=High, 26=Low	
UART 0:	10	
UART 1:	11	
8PI:	12	
GPIO A:	13	
GPIO B:	14	
GPIO C:	15	
GPIO D:	16	
GPIO E:	17	
com.ajile.drivers irq.InterruptController gpioEInterrupt0V	18	

Runtime: Runtime_cldc Configuration: aj100LCDConfiguration

2406

Fig. 24

JemBuilder: C:\local\ServerSocket_and_GPIO.apj

File Edit Project Tools Help

Project

- Output Files
- Memory
- PLL Setup
- Pin Setup1
- Pin Setup2
- Chip Selects
- JVM0**
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aJ100 Interrupts
 - Invoke**
 - Traps
 - Patch Classes
 - More Classes
- Drivers
- JVM1**
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aJ100 Interrupts
 - Invoke
 - Traps
 - Patch Classes
 - More Classes
- Build Output

Implicit Invoke Handlers

Fully Qualified Method Name Including Signature
eg: com.ajlle.UartHandlerQV
Note: Must be a static non-synchronized method

multiArray(II)Ljava/lang/Object;

Invoke 0: com.ajlle.jem.RTS_Memory.invokeMultiArrayQ(Ljava/lang/Object;

resolveClassRef(II)V

Invoke 1: com.ajlle.jem.RTS_DLink.invokeResolveClassRef(II)V

resolveInstanceFieldRef(II)V

Invoke 2: com.ajlle.jem.RTS_DLink.invokeResolveInstanceFieldRef(II)V

resolveClassFieldRef(II)V

Invoke 3: com.ajlle.jem.RTS_DLink.invokeResolveClassFieldRef(II)V

resolveInstanceMethodRef(II)V

Invoke 4: com.ajlle.jem.RTS_DLink.invokeResolveInstanceMethodRef(II)V

resolveClassMethodRef(II)V

Invoke 5: com.ajlle.jem.RTS_DLink.invokeResolveClassMethodRef(II)V

resolveInterfaceMethodRef(II)V 2506

Invoke 6: com.ajlle.jem.RTS_DLink.invokeResolveInterfaceMethodRef(II)V

stackChunkCopy(II)V

Invoke 7: com.ajlle.jem.RTS_Memory.invokeStackChunkCopy(II)V 2502

throwID(II)V

Invoke 8: com.ajlle.jem.RTS_Exception.invokeThrowID(II)V

aThrow(II)Ljava/lang/Throwable;II)V

Invoke 9: com.ajlle.jem.RTS_Exception.invokeThrowObject(II)Ljava/lang/Throwable;II)V 2504

2500

Runtime: Runtime_cldc Configuration: aJ100LCDConfiguration

Page 56 of 62

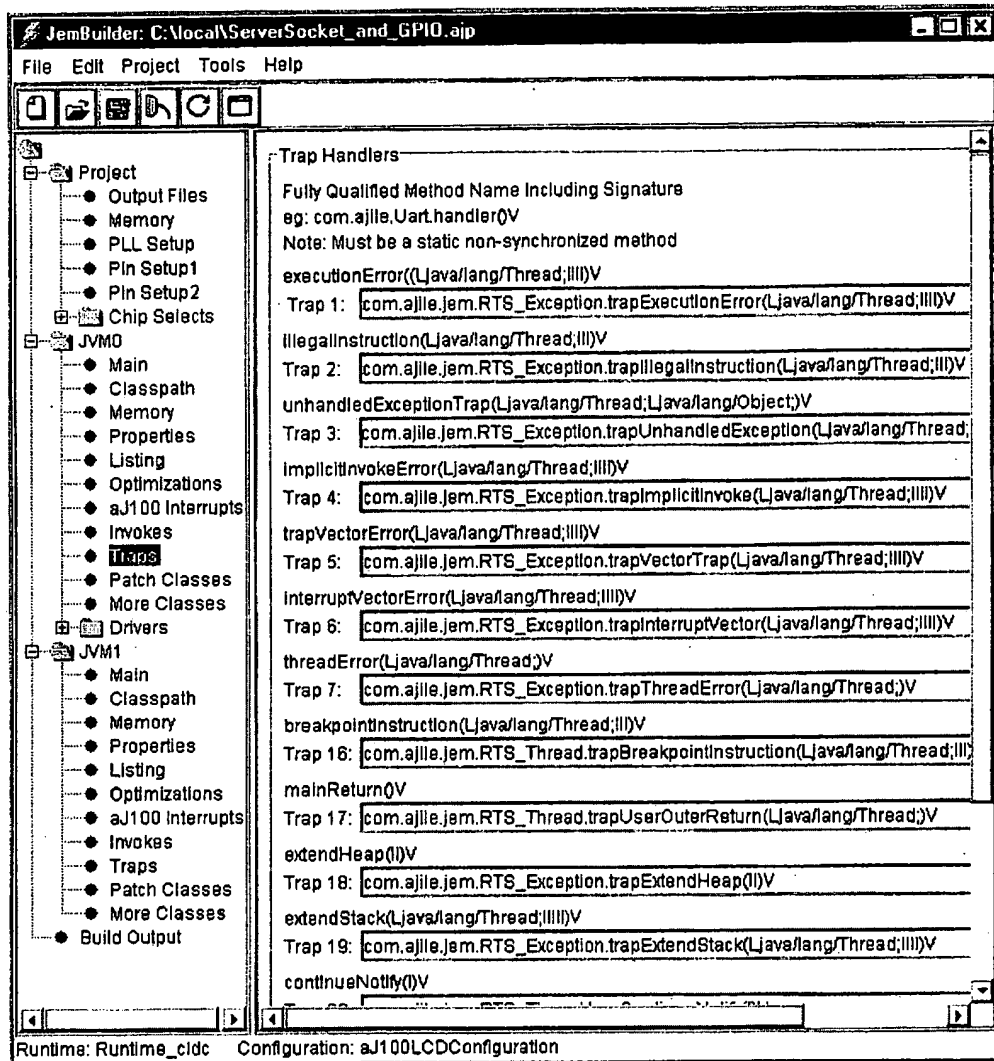


Fig. 26

1060200123242550

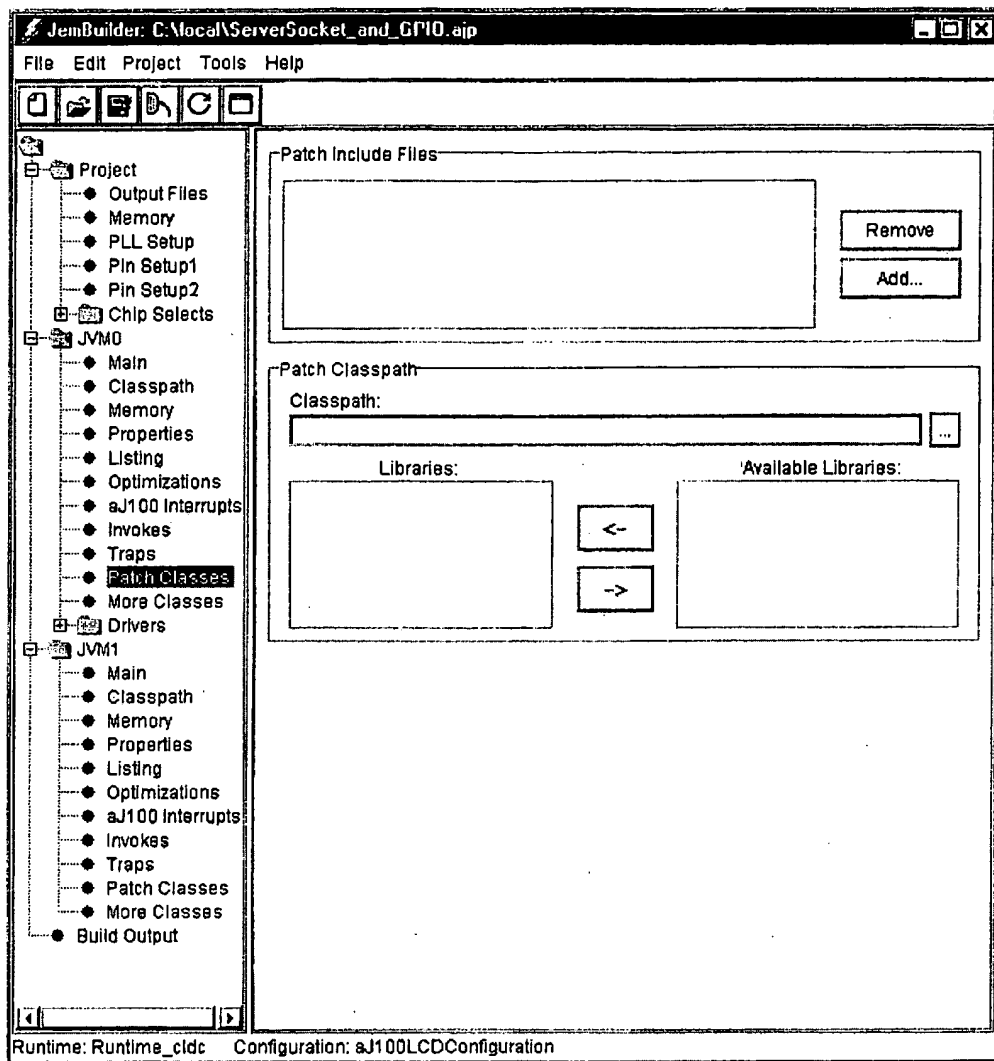


Fig. 27

Fig. 28

09561360 09561360

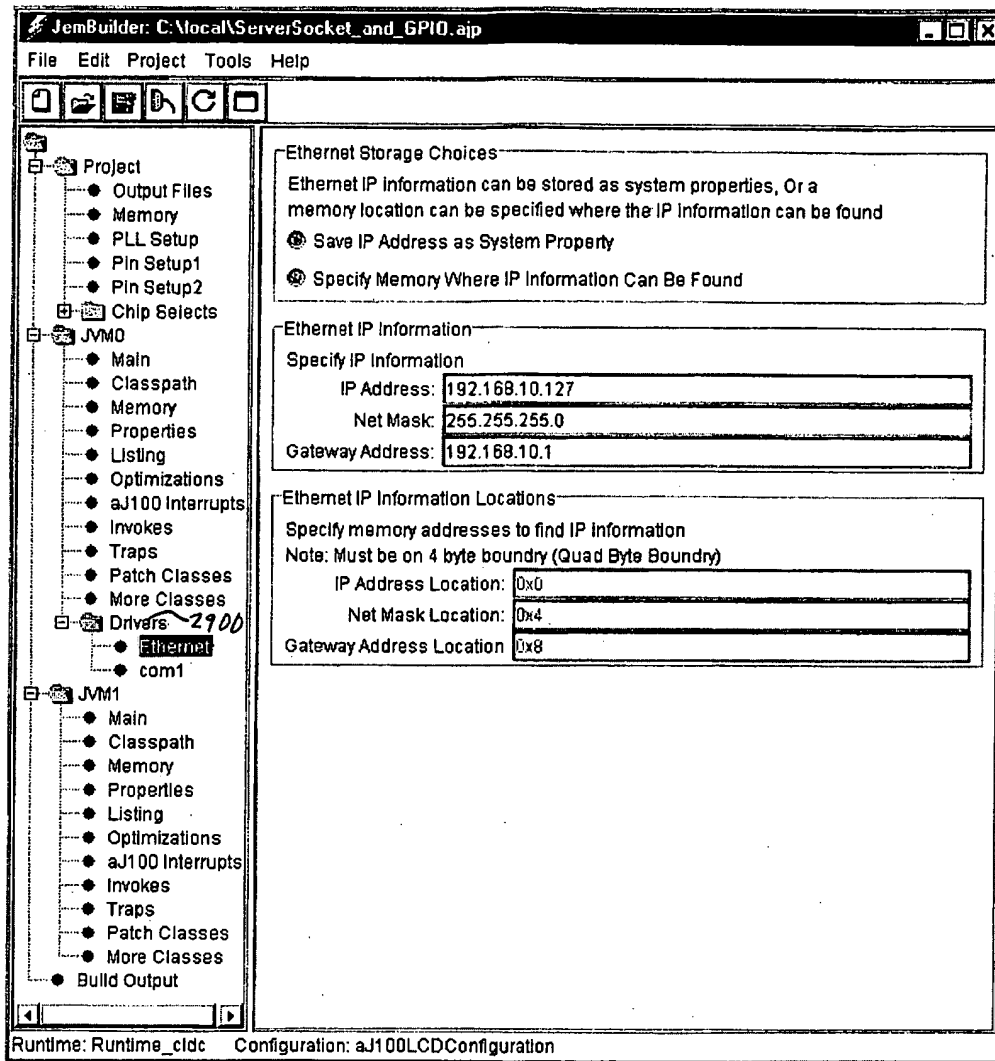


Fig. 29

The screenshot shows the JemBuilder IDE interface. On the left is a project tree with the following structure:

- Project
 - Output Files
 - Memory
 - PLL Setup
 - Pin Setup1
 - Pin Setup2
 - Chip Selects
- JVM0
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aJ100 Interrupts
 - Invokes
 - Traps
 - Patch Classes
 - More Classes
- Drivers
 - Ethernet
 - com1
- JVM1
 - Main
 - Classpath
 - Memory
 - Properties
 - Listing
 - Optimizations
 - aJ100 Interrupts**
 - Invokes
 - Traps
 - Patch Classes
 - More Classes
 - Build Output

On the right is the 'Interrupt Handlers' configuration window. It contains the following sections and fields:

- Fully Qualified Method Name Including Signature**
eg: com.ajile.Uart.handlerQV
Note: Must be a static non-synchronized method with QV signature
- Transfer Error XERR**
- IRQ 0:** [Text Field]
- Power Down Warning PDW**
- IRQ 1:** [Text Field]
- External NMI**
- IRQ 3:** [Text Field]
- Arithmetic Error (JVM aware)**
- IRQ 5:** [Text Field]
- Timer/Counter TCO**
- IRQ 7:** [Text Field]
- Peripheral Interrupts**

	Priority:
Timer:	10=High, 26=Low
UART 0:	10
UART 1:	11
SPI:	12
GPIO A:	13
com.ajile.drivers irq.InterruptController.gpioAInterruptQV	14
GPIO B:	15
GPIO C:	16

At the bottom of the window, the status bar shows: Runtime: Runtime_cldc Configuration: aJ100LCDConfiguration

Fig. 31